Programming with Blocks and Grand Central Dispatch

Session 406

These are confidential sessions—please refrain from streaming, blogging, or taking pictures.

Programming with Blocks and Grand Central Dispatch

Blaine Garst
Wizard of Runtimes

Dave Zarzycki
Multicore Maestro

Blocks

```
(lambda (a) (add a d))
```

Scheme

LISP

SmallTalk

```
10 timesRepeat:[pen turn:d; draw]
```

Ruby

```
z.each {|val| puts(val + d.to_s) }
```

```
(lambda (a) (add a d))
```

Scheme

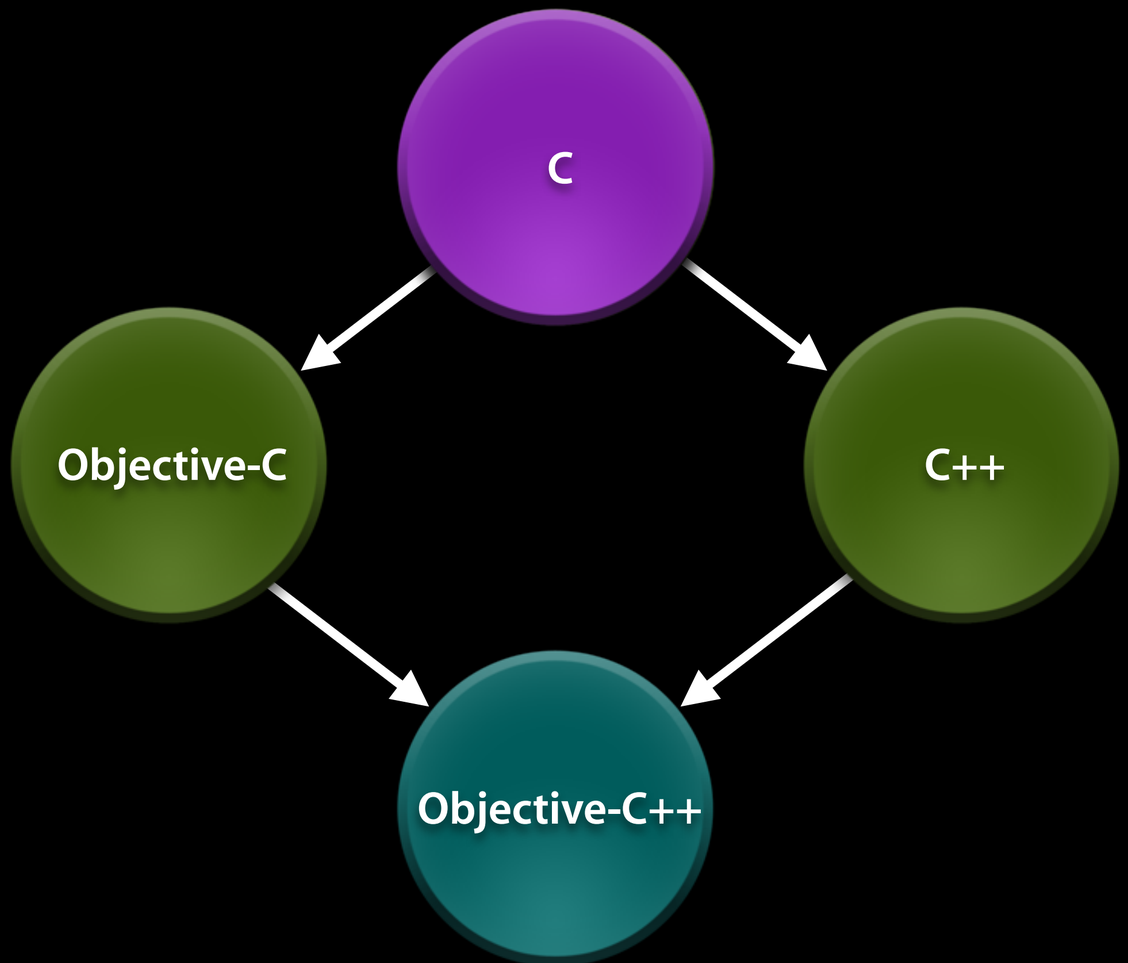
LISP

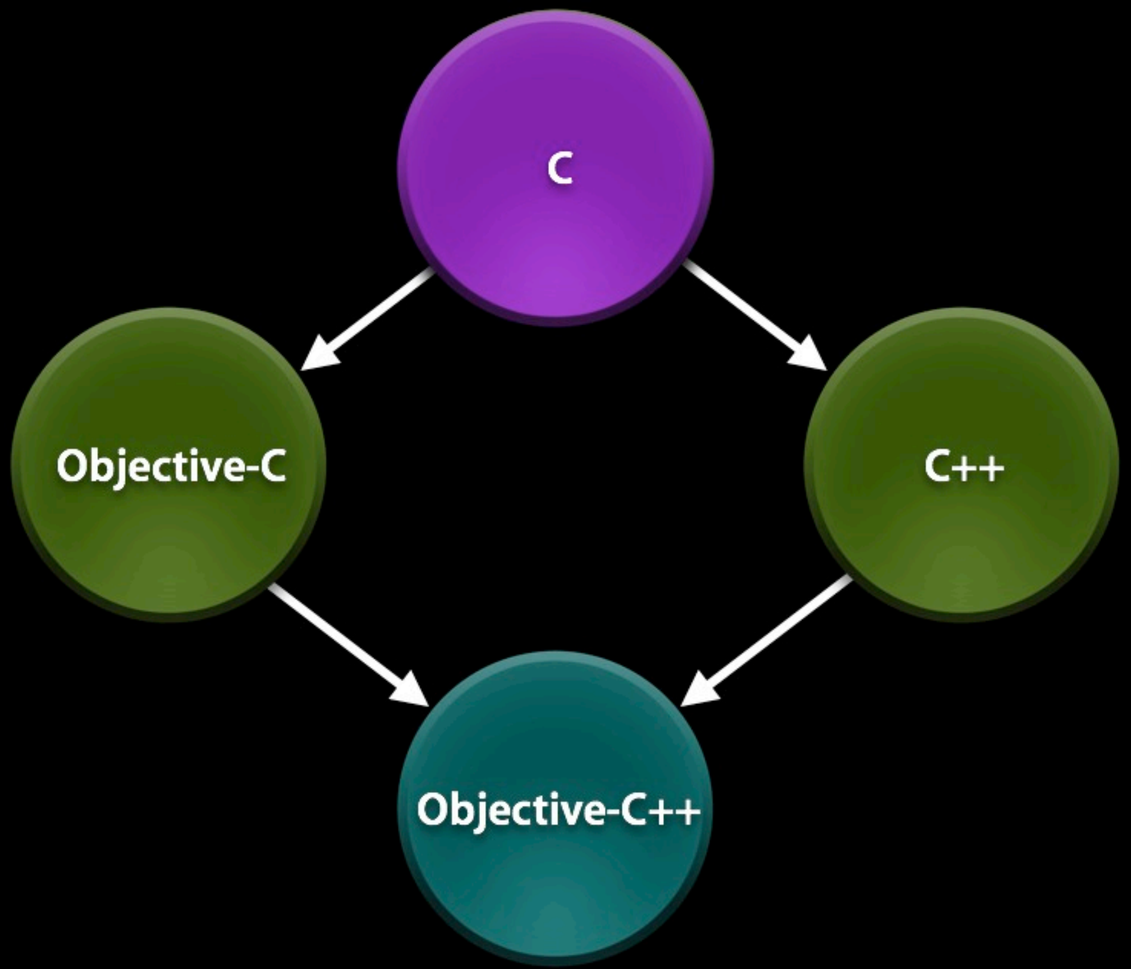
SmallTalk

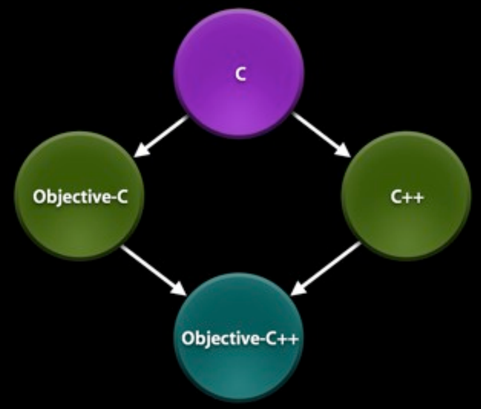
```
10 timesRepeat:[pen turn:d; draw]
```

Ruby

```
z.each {|val| puts(val + d.to_s) }
```


C
Objective-C
C++
Objective-C++


C
Objective-C
C++
Objective-C++




```
repeat(10, ^{ putc('0'+d); });
```

```
int d = i % 10;
repeat(10, ^{ putc('0'+d); });
```

A Block Reference Declaration

```
void (^block)(void);
```

Use Typedefs

```
typedef void (^workBlk_t)(void);
```

Do Block N Times Function

```
typedef void (^workBlk_t)(void);

void
repeat(int n, workBlk_t block) {
        for (int i = 0; i < n; ++i)
                block();
}
```

Traditional Uses for Closures…

Wrapper

```
typedef void (^vstr_t)(char *);
void AllLinesInFile(char *f, vstr_t block)
{
    FILE *fp = fopen(f, "r");
    if (!fp) return;
    char line[1024];
    while (fgets(line, 1024, fp))
        block(line);
    fclose(fp);
}
```

Wrapper

```
think;
think;
think;
AllLinesInFile(filename,  ^ (char *line) {
    compute;
    compute;
    compute;
}
```

Non-Wrapper Version

```
think;
think;
think;
FILE *fp = fopen(filename, "r");
if (fp) {
    char line[1024];
    while (fgets(line, 1024, fp)) {
        compute;
        compute;
        compute;
    }
    fclose(fp);
}
```

Wrapper

```
think;
think;
think;
AllLinesInFile(filename,  ^ (char *line) {
    compute;
    compute;
    compute;
}
```

Map

```
@implementation NSArray (WWDC)
- (NSArray *)map:(id (^)(id))xform {
    id result = [NSMutableArray array];
    for (id elem in self)
        [result addObject:xform(elem)];
    return result;
}
@end
```

Collect

```
@implementation NSArray (WWDC)
- (NSArray *)collect:(BOOL (^)(id))predicate {
    id result = [NSMutableArray array];
    for (id elem in self)
        if (predicate(elem))
            [result addObject:elem];
    return result;
}
@end
```

"Collect" in Use

```
id longLines = [allLines collect:
                        ^ BOOL (id item) {
        return [item length] > 20;
}];
```

New Uses

Traditional: qsort_r

```
void qsort_r (
        void *array,
        size_t nElems,
        size_t width,
        void *callback_argument,
        int (*compare)(void *l, void *r, void *cb)
);
```

Traditional: qsort_r

1. Declare custom structure type

```
typedef struct {
    int stringOption;
    BOOL ageFirst;
    ...;
} PeopleSort_t;
```

Traditional: qsort_r

2. Write compare function using custom structure

```
int myCustomCompare(void *p1, void *p2,
                    void *ctx) {
    ...
    if (((PeopleSort_t *)ctx)->ageFirst)
        ...
    ...
}
```

Traditional: qsort_r

3. Declare and initialize custom structure

```
void sortPeople(BOOL ageFirst) {
    int options = ...
    PeopleSort_t ctx = {
        ageFirst,
        options,
        ...};
```

Traditional: qsort_r

4. Call qsort_r

```
...
qsort_r(array, nelems, width,
                        &ctx, myCustomCompare);
```

New: qsort_b

```
void qsort_b (
        void *array,
        size_t nElems,
        size_t width,
        int (^compare)(void *l, void *r)
);
```

New: qsort_b

1. call qsort_b

```
void sortPeople(BOOL ageFirst) {
    int options = ...
qsort_b(array, nelems, width, ^ {
        ...
        if (ageFirst > 0)
            ...;
        ...
 });
```

Blocks are a Far Better Call Back API

- Old style
  - `qsort_r(..., void *ctx, cb_t func);`
  - `performSelector:withObject:`
  - `performSelector:withObject:withObject:`
  - `setTarget:, setAction: pair`
  - NSNotifications require userInfo dictionary to hold requisite data
- New style
  - `setCallback(^(param-list) { code });`
  - Language interoperable!

Lifetime

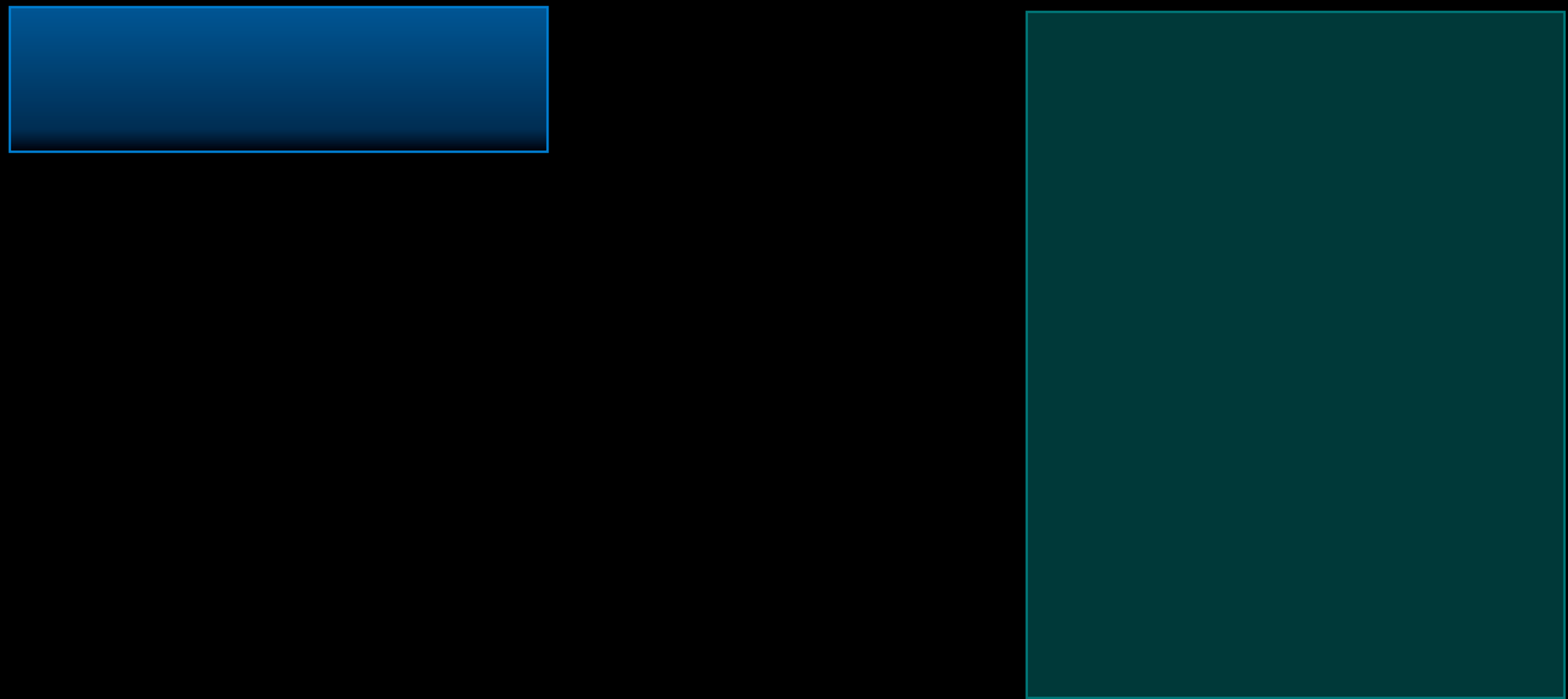
- Blocks start out on the stack
- Can be copied to the heap
- Blocks have private const copy of stack (auto) variables
- Mutable variables must be declared with a `__block` keyword
  - Shared with all other blocks that use that variable
  - Shared with the scope of declaration

New __block Storage Class

```
__block id result = nil;
[dict enumerateKeysAndObjectsUsingBlock:
      ^ (id key, id value, BOOL *stop) {
            if ([object isEqual:sought)]) {
                result = key;
                *stop = YES;
            }
        }];
```

Block Lifetime

Initial State Before Function Call

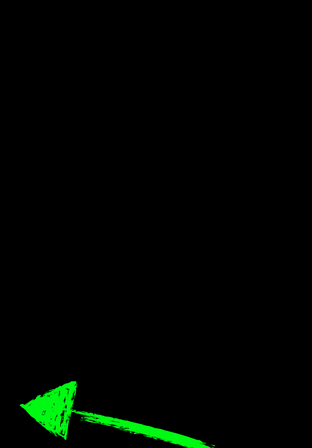


Stack

Heap

State Before Copy to Heap

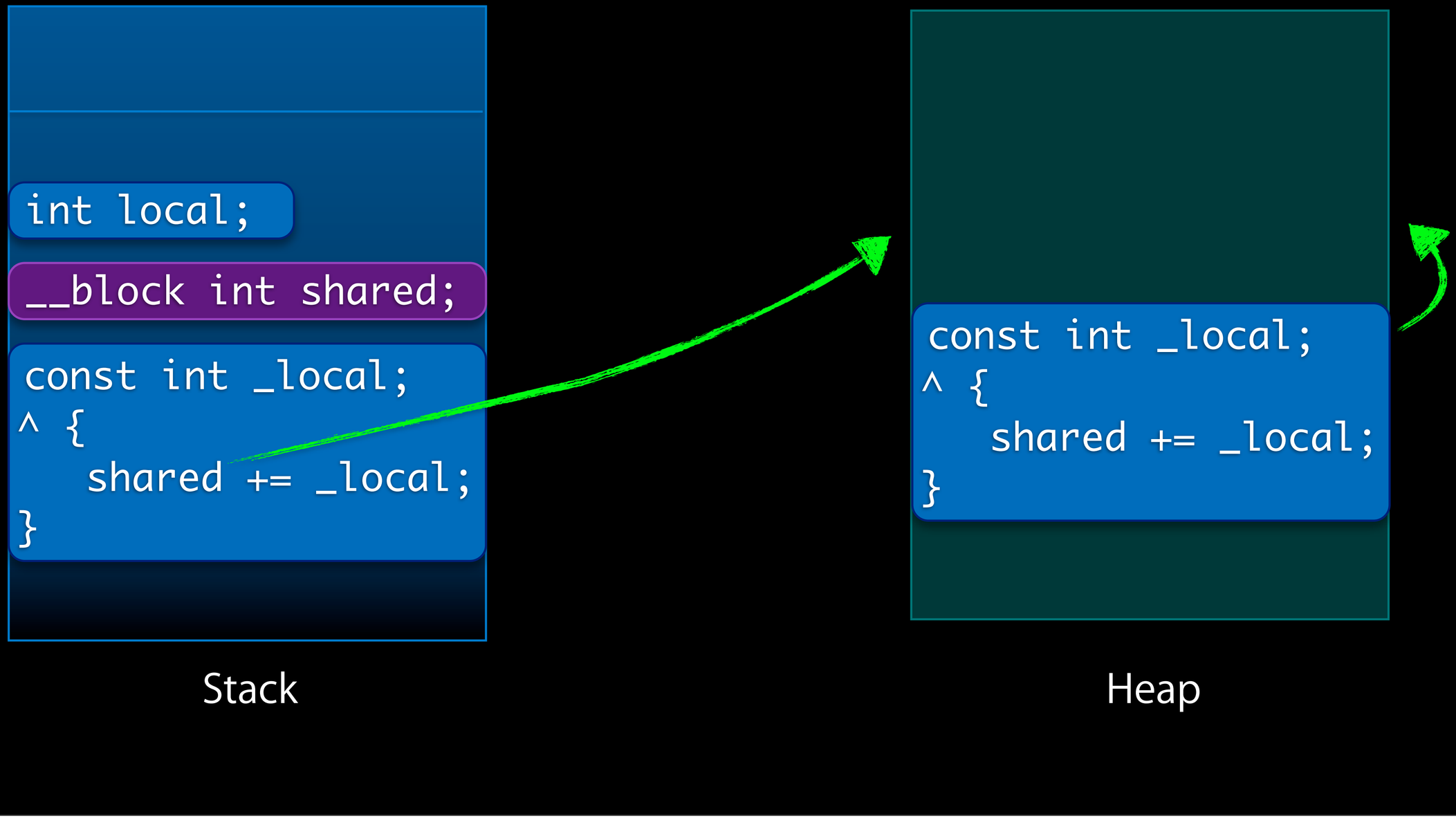
```
int local;

__block int shared;

const int _local;
^ {
    shared += _local;
}
```



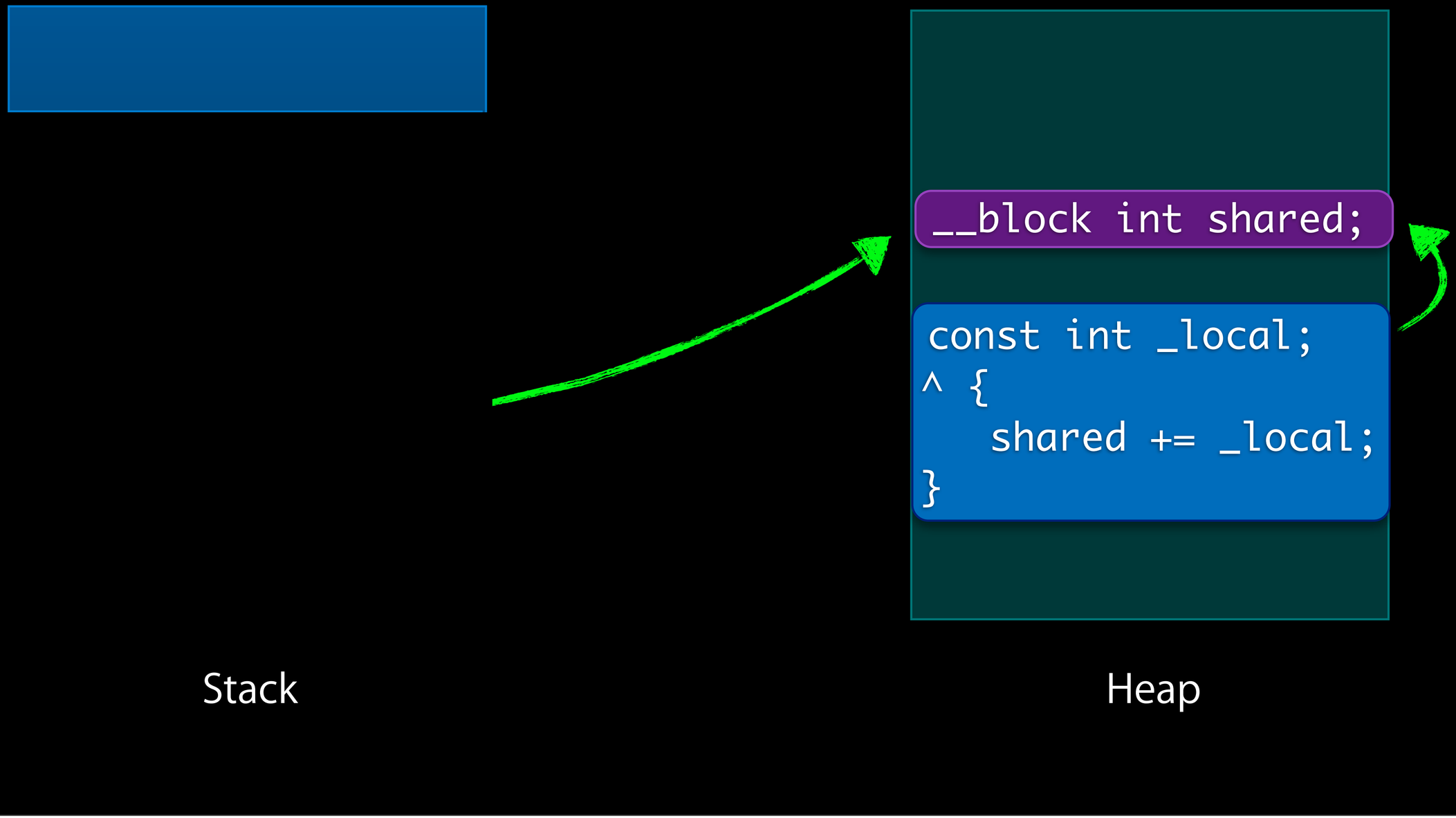
Stack

Heap

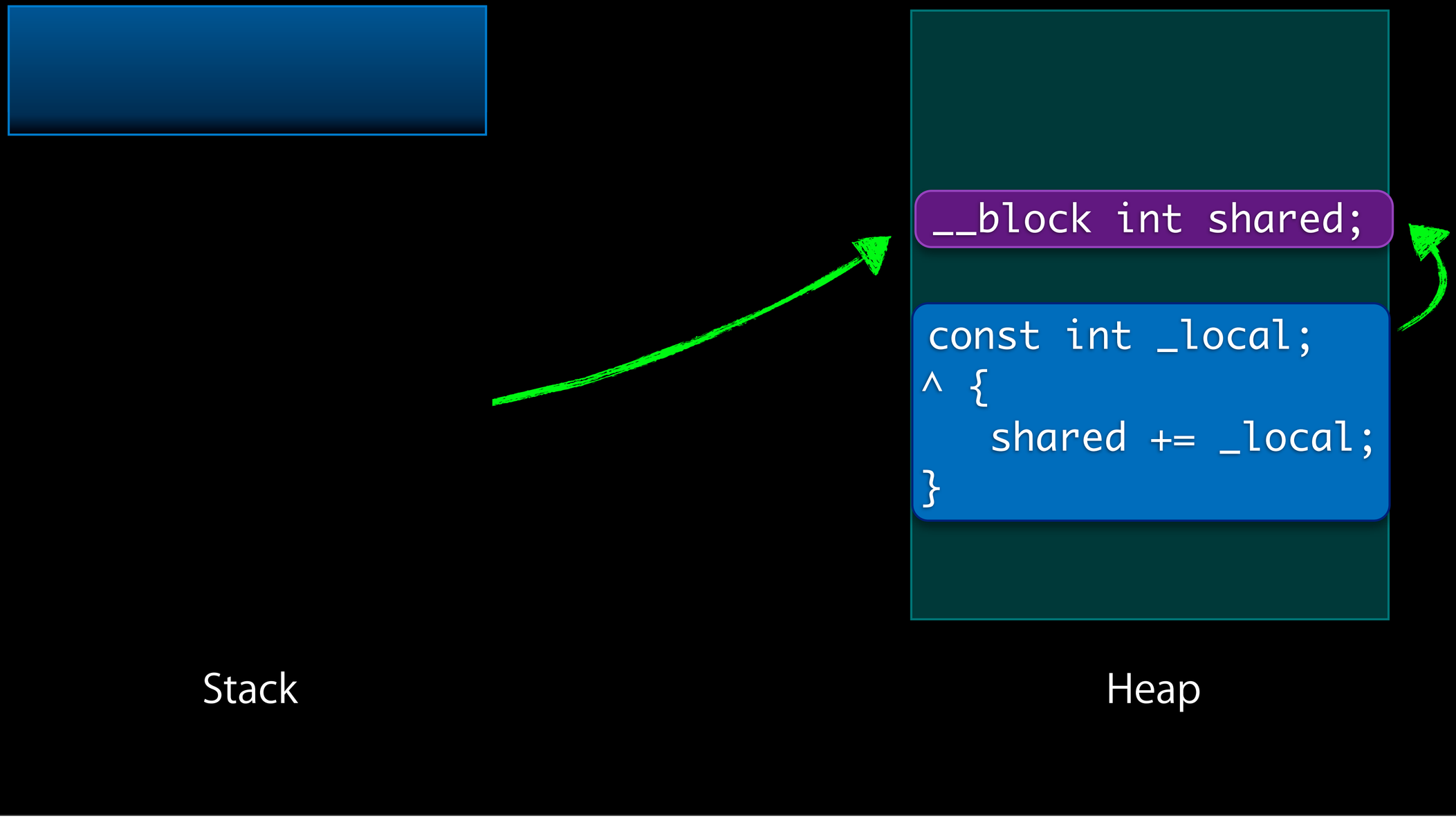
After Copy to Heap



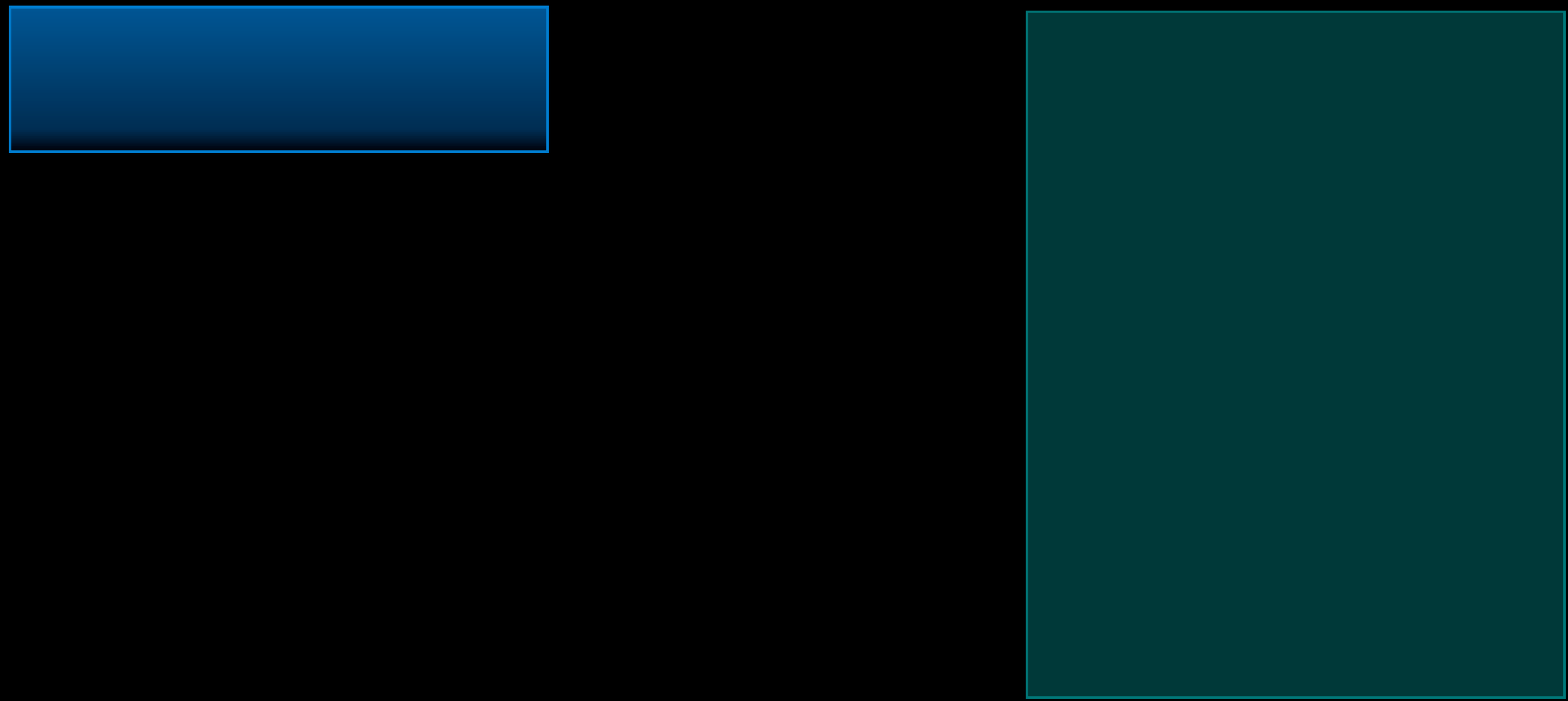
Case 1: Function Might Return…



Case 2: Heap Object Might Perish First



Leaving Us at Our Initial State



Stack

Heap

Objective-C Specializations

- All blocks are also always Objective-C objects!
  - Respond to the `-copy` (and `-release`) messages
  - `@property(copy) void (^graphicCallBack)(CGImage x);`

Objective-C and Garbage Collection Advancements	Russian Hill Wednesday 5:00PM

Summary: What are Blocks Good For?

- Over 100 Snow Leopard APIs use blocks!
- Traditional (blocking, single-threaded) uses
  - Collection iterations with or without options
  - Collection map, reduce, collect, …
  - Wrappers
- Call back replacements (various-threaded-nesses)
- Asynchronous (nonblocking, multithreaded) uses
  - Concurrent computation on independent data
  - Serialized computation on shared data
  - All combos of the above

GCD

GCD?

Asynchronous Blocks

```
dispatch_async(queue, ^{
    printf(“Hello\n”);
});
```

16 Virtual Cores

2 Cores

100% Multicore

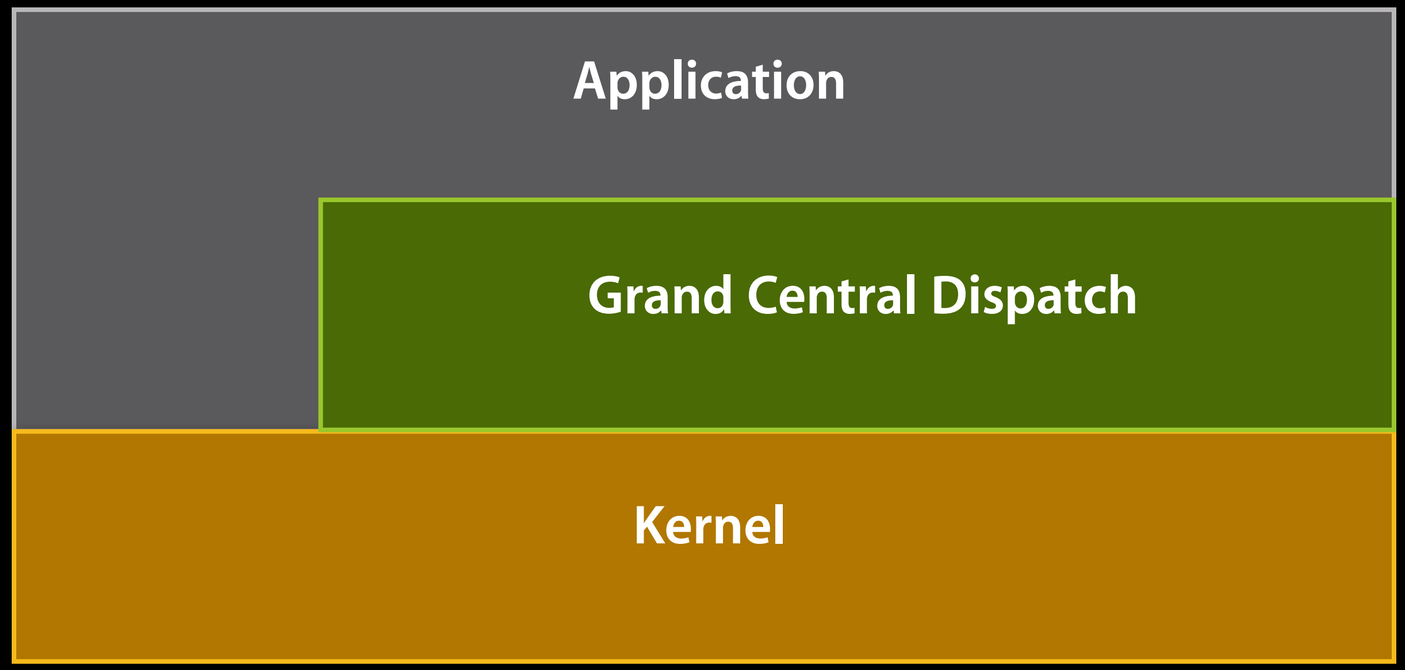
One More Problem

User experience

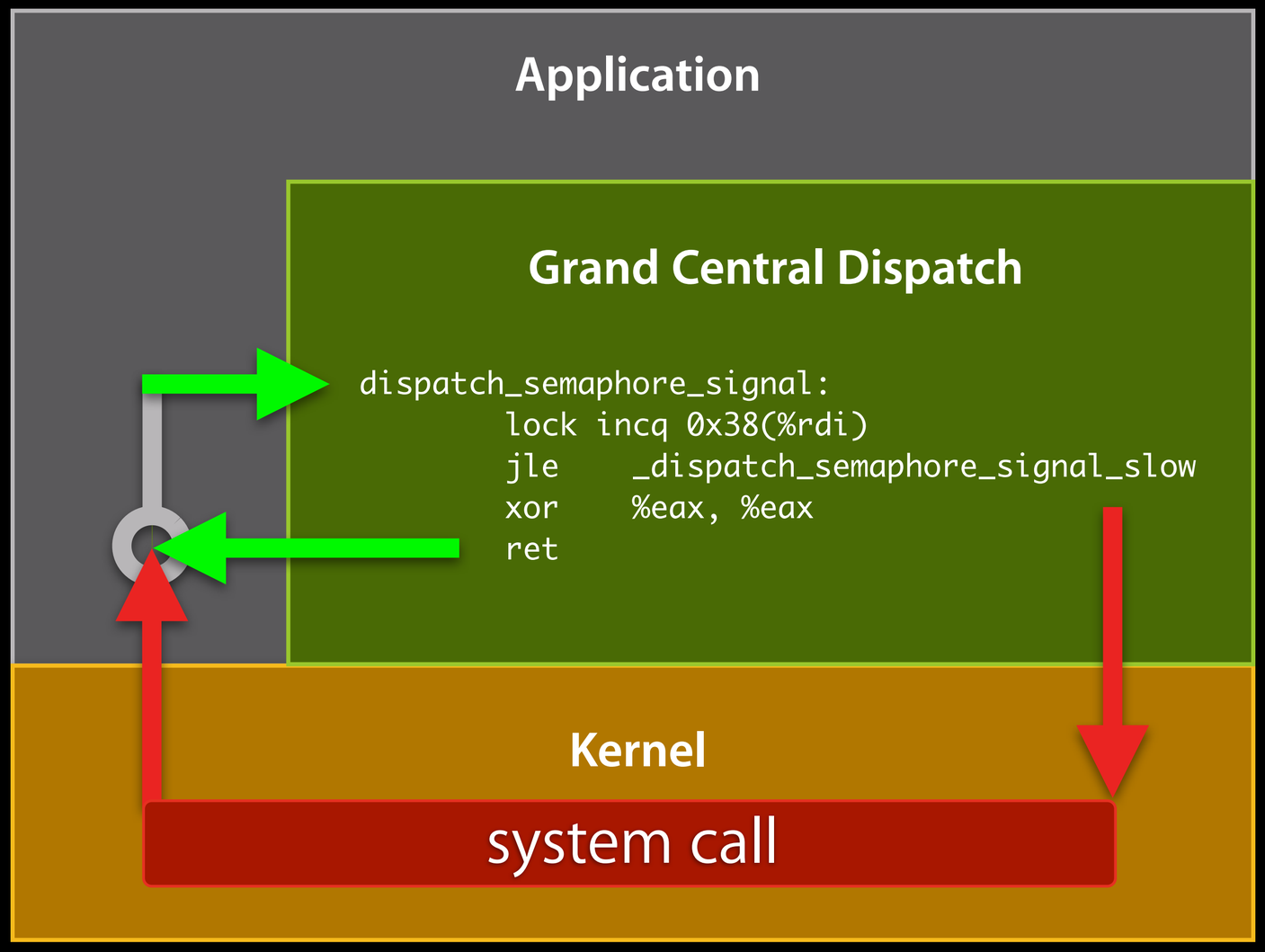
GCD?

Fast ***Easy*** ***Fun***

Technology Stack



Kernel Bypass



200x

Dispatch semaphore versus pthread condition variables under contention on a 2.26GHz 16-way Mac Pro

Fast ***Easy*** ***Fun***

Better Metaphor

Blocks and Queues

```
dispatch_async(queue, ^{
    printf(“Hello\n”);
});
```

Fast ***Easy*** ***Fun***

Stories on Campus

“Wow, my subsystem runs a lot faster!”

“Why hasn’t it always been this way?”

“Dispatch is like writing poetry.”

Technology

Overview

Technology Overview

- GCD is part of libSystem.dylib
- Available to all applications
  - `#include <dispatch/dispatch.h>`

Queues
dispatch_queue_t

dispatch_queue_create
dispatch_queue_get_label
dispatch_get_main_queue
dispatch_get_global_queue
dispatch_get_current_queue
dispatch_main
dispatch_apply
dispatch_apply_f
dispatch_async
dispatch_async_f
dispatch_sync
dispatch_sync_f
dispatch_after
dispatch_after_f

Semaphores
dispatch_semaphore_t

dispatch_semaphore_create
dispatch_semaphore_signal
dispatch_semaphore_wait

Objects
dispatch_object_t

dispatch_retain
dispatch_release
dispatch_suspend
dispatch_resume
dispatch_debug
dispatch_get_context
dispatch_set_context
dispatch_set_finalizer_f
dispatch_set_target_queue

Groups
dispatch_group_t

dispatch_group_create
dispatch_group_enter
dispatch_group_leave
dispatch_group_wait
dispatch_group_notify
dispatch_group_notify_f
dispatch_group_async
dispatch_group_async_f

Sources
dispatch_source_t

dispatch_source_create
dispatch_source_cancel
dispatch_source_testcancel
dispatch_source_merge_data
dispatch_source_get_handle
dispatch_source_get_mask
dispatch_source_get_data
dispatch_source_set_timer
dispatch_source_set_event_handler
dispatch_source_set_event_handler_f
dispatch_source_set_cancel_handler
dispatch_source_set_cancel_handler_f

Not Objects

Time
dispatch_time_t

dispatch_time
dispatch_walltime

Once
dispatch_once_t

dispatch_once
dispatch_once_f

Queues

dispatch_queue_t

dispatch_queue_create
dispatch_queue_get_label
dispatch_get_main_queue
dispatch_get_global_queue
dispatch_get_current_queue
dispatch_main
dispatch_apply
dispatch_apply_f
dispatch_async
dispatch_async_f
dispatch_sync
dispatch_sync_f
dispatch_after
dispatch_after_f

Objects

dispatch_object_t

dispatch_retain
dispatch_release
dispatch_suspend
dispatch_resume
dispatch_debug
dispatch_get_context
dispatch_set_context
dispatch_set_target_queue
dispatch_set_finalizer_f

Sources

dispatch_source_t

dispatch_source_create
dispatch_source_cancel
dispatch_source_testcancel
dispatch_source_merge_data
dispatch_source_get_handle
dispatch_source_get_mask
dispatch_source_get_data
dispatch_source_set_timer
dispatch_source_set_event_handler
dispatch_source_set_event_handler_f
dispatch_source_set_cancel_handler
dispatch_source_set_cancel_handler_f

Semaphores

dispatch_semaphore_t

dispatch_semaphore_create
dispatch_semaphore_signal
dispatch_semaphore_wait

Groups

dispatch_group_t

dispatch_group_create
dispatch_group_enter
dispatch_group_leave
dispatch_group_wait
dispatch_group_notify
dispatch_group_notify_f
dispatch_group_async
dispatch_group_async_f

Not Objects

Time

dispatch_time_t

dispatch_time
dispatch_walltime

Once

dispatch_once_t

dispatch_once
dispatch_once_f

Related Sessions

Understanding Grand Central Dispatch in Depth	Presidio Thursday 9:00AM
Migrating Your Application to Grand Central Dispatch	Pacific Heights Thursday 2:00PM
Designing Your Cocoa Application for Concurrency	Nob Hill Tuesday 3:30PM
What's New in Instruments	Pacific Heights Wednesday 3:30PM
Objective-C and Garbage Collection Advancements	Russian Hill Wednesday 5:00PM
Advanced Debugging and Performance Analysis	Nob Hill Thursday 9:00AM

GCD Objects

GCD Objects

- Some polymorphic API
- Reference counted memory management
  - `dispatch_retain(object);`
  - `dispatch_release(object);`

- GCD does the right thing
  - Retains parameters when needed

- A few other shared functions

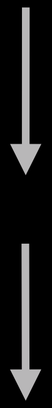
Queues

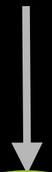
Queues

- Lightweight list of blocks
- Asynchronous execution
- Enqueue/dequeue is FIFO

Your Thread

A = ^{ ... };





Dispatch Queue

Automatic Thread

Queues

```
dispatch_async(queue, ^{ /* A */ });

dispatch_async(queue, ^{ /* B */ });

dispatch_async(queue, ^{ /* C */ });
```

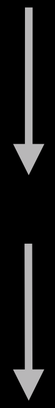
Global Queues

Global Queues

- Lightweight list of blocks
- Asynchronous execution
- Enqueue/dequeue is FIFO
- Concurrent execution

Your Thread

A = ^{ ... };



Global Dispatch Queue

Automatic Thread

Automatic Thread

Queue of Blocks

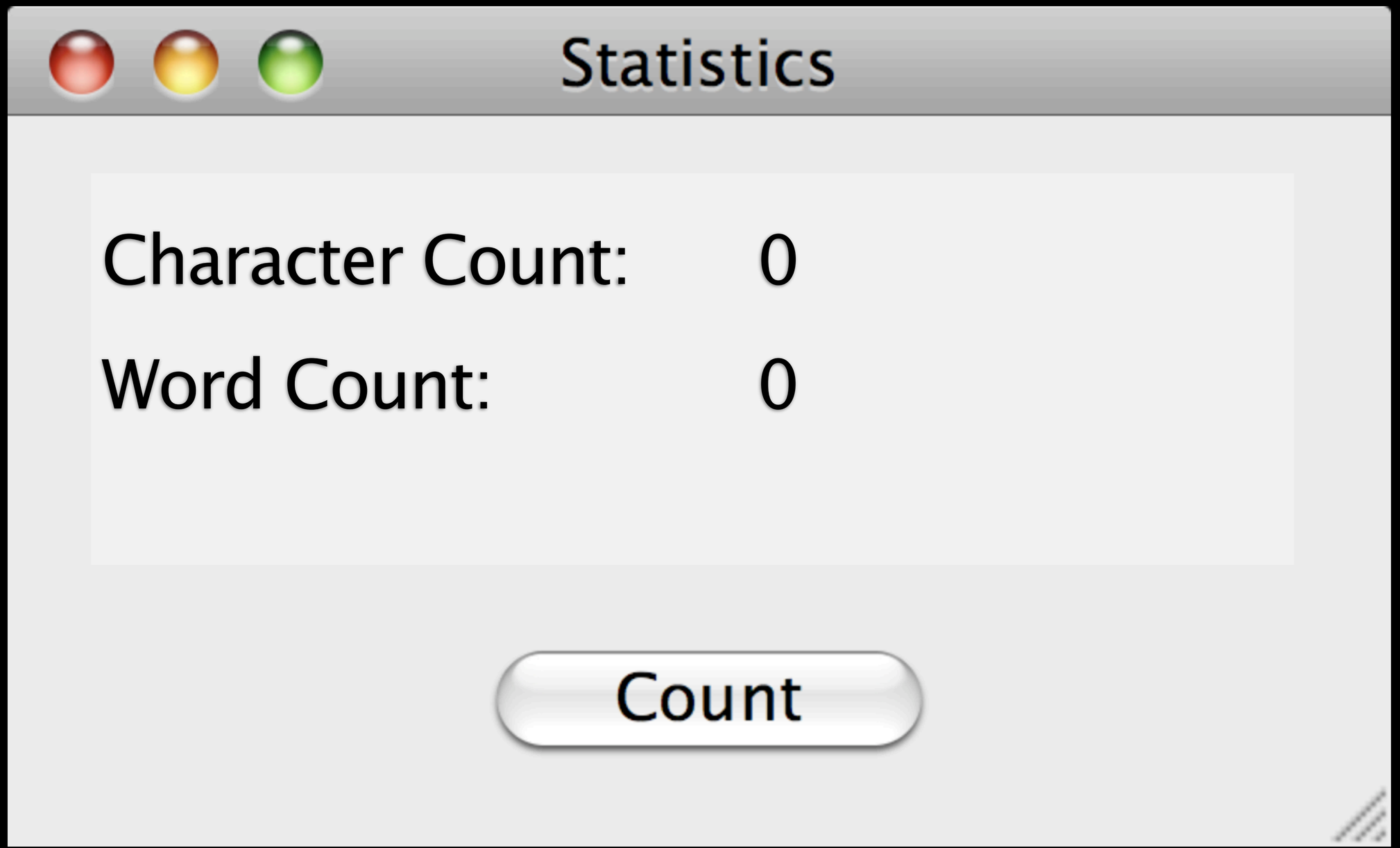
```
queue = dispatch_get_global_queue(0, 0);

dispatch_async(queue, ^{ /* A */ });

dispatch_async(queue, ^{ /* B */ });

dispatch_async(queue, ^{ /* C */ });
```

Easy Concurrency

Suppose you wanted to count the characters, words, and paragraphs in a document.

And suppose that your document is large,

very large…

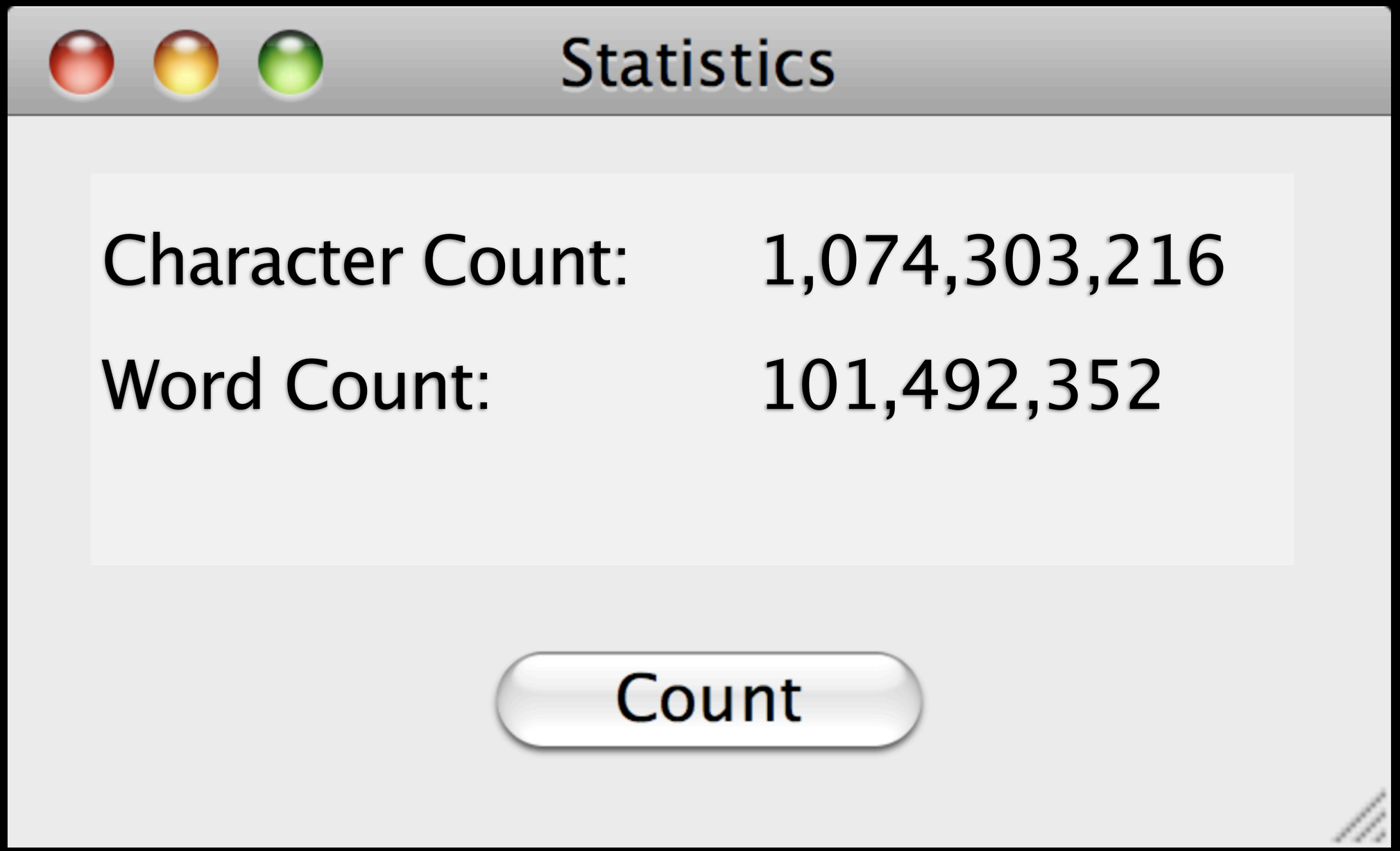

Statistics
Character Count: 0
Word Count: 0
Count

Statistics

Character Count: 0

Word Count: 0

Count


Statistics
Character Count: 1,074,303,216
Word Count: 101,492,352
Count

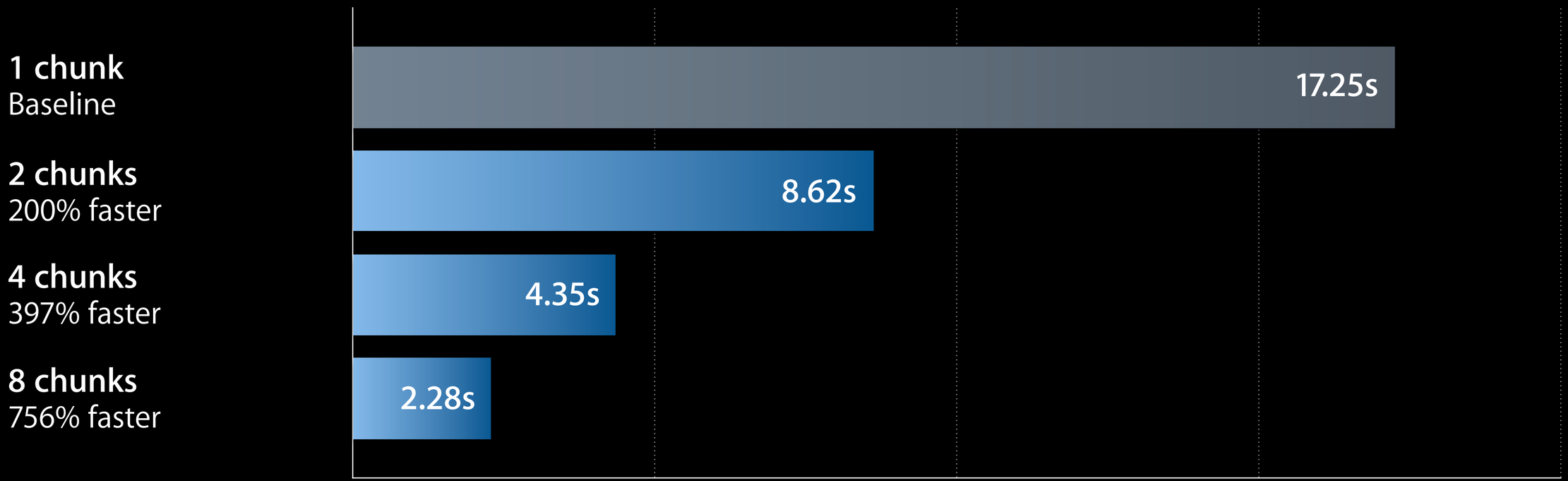
```
for (i = 0; i < count; i++) {
    results[idx] = do_scan(text, idx);
}

total = summarize(results, count);
```

```
                                {
        results[idx] = do_scan(text, idx);
    }

    total = summarize(results, count);
```

```
queue = dispatch_get_global_queue(0, 0);

dispatch_apply(count, queue, ^(size_t idx) {
    results[idx] = do_scan(text, idx);
});

total = summarize(results, count);
```

dispatch_apply() Elapsed Time

Smaller bars are better



Elapsed time processing an approx. 1GB text file on an 8 core Mac Pro

Asynchronous Design

Asynchronous Design

```
- (IBAction)compute:(NSButton *)sender
{
    NSDictionary *dict = [myDoc summarize];
    [myModel setDict:dict];
    [myView setNeedsDisplay:YES];
    [dict release];
}
```

Asynchronous Design

```
- (IBAction)compute:(NSButton *)sender
{
    dispatch_async(dispatch_get_global_queue(0, 0), ^{
        NSDictionary *dict = [myDoc summarize];
        dispatch_async(dispatch_get_main_queue(), ^{
            [myModel setDict:dict];
            [myView setNeedsDisplay:YES];
            [dict release];
        });
    });
}
```

Main Queue

Dispatch Main Queue

- Executes blocks serially on main thread

- Cooperates with the Cocoa main run loop
  - NSApplicationMain()

- Usable by pure GCD programs
  - dispatch_main()

Main Queue

```
int main(int argc, char **argv)
{
    dispatch_queue_t queue = dispatch_get_main_queue();

    dispatch_async(queue, ^{
        printf(“%s: Hello World\n”, argv[0]);
    });

    // Application and library initialization
    // All may use the main queue

    NSApplicationMain(argc, argv);      // Cocoa apps
    // dispatch_main();                 // Pure GCD apps
}
```

Serial Queues

Serial Queues

- Lightweight list of blocks
- Asynchronous execution
- Enqueue/dequeue is FIFO
- Serial completion

Your Thread

B = ^{ ... };

Another Thread

C = ^{ ... };

Dispatch Queue

Automatic Thread

Serial Queues

// Thread One	// Thread Two
dispatch_async(queue, ^{ /* A */ });	

dispatch_async(queue, ^{
    /* B */
});
	dispatch_async(queue, ^{
	    /* C */
	});

Serial Queues

```
queue = dispatch_queue_create(“com.example.purpose”, NULL);
```

```
label = dispatch_queue_get_label(queue);
```

```
printf(“Queue %p: %s\n”, queue, label);
```

Multiple Serial Queues

Multiple Serial Queues

- Queues are lightweight
- No imposed limits
- Natural to assign one queue per task
- Potential for inter-task concurrency

Your Thread

E = ^{ ... };

Dispatch Queue

Dispatch Queue

Automatic Thread

Multiple Serial Queues

```
q1 = dispatch_queue_create(“com.example.task1”, NULL);
q2 = dispatch_queue_create(“com.example.task2”, NULL);

dispatch_async(q1, ^{ /* A */ });

dispatch_async(q2, ^{ /* B */ });

dispatch_async(q1, ^{ /* C */ });

dispatch_async(q2, ^{ /* D */ });

dispatch_async(q1, ^{ /* E */ });
```

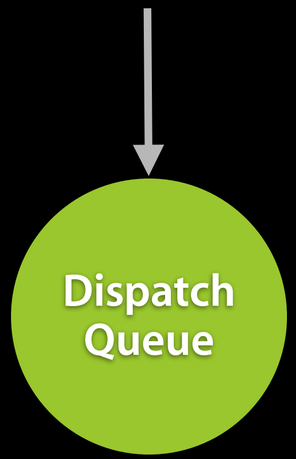
Grouping Multiple Blocks

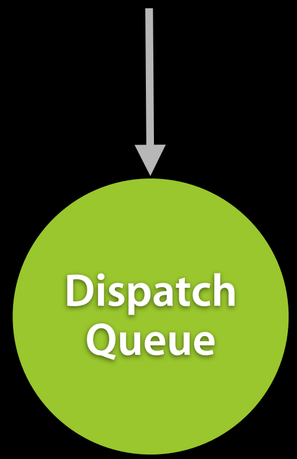
Grouping Blocks

- Track multiple blocks as a group
  - Empty group notification
- Blocks may run on different queues
- Blocks may add more blocks to the group
  - “Recursive decomposition”

Your Thread

E = ^{ ... };





Automatic Thread

Grouping Blocks

```
dispatch_group_t group = dispatch_group_create();
```

```
for (i = 0; i < count; i++) {
    dispatch_group_async(group, tasks[i]->queue, ^{
        do_something(tasks[i]);
    });
}
```

```
dispatch_group_notify(group, dispatch_get_main_queue(), ^{
    do_something_else(data);
});
```

```
dispatch_release(group);
```

Completion

- Groups are a generic completion notification system
- Nested blocks are the simple case
  - Fewer lines of code
  - More efficient

Completion

Nested blocks

```
dispatch_async(dispatch_get_global_queue(0, 0), ^{
    result_t rval = do_something();
    dispatch_async(dispatch_get_main_queue(), ^{
        result_draw(rval);
        result_free(rval);
    });
});
```

Completion

- What if asynchronous notifications are not practical?
- Code must then wait

Waiting

Waiting

- Bridging synchronous and asynchronous code
- Waiting for blocks to complete
  - Fork/join model
- Wait for one block
  - dispatch_sync()
- Groups are a generic solution

Waiting

```
__block result_t rval;

dispatch_sync(task->queue, ^{
    rval = do_something(task);
});

// use rval
```

Waiting

```
dispatch_group_t group = dispatch_group_create();
__block result_t rvals[count];
```

```
for (i = 0; i < count; i++) {
    dispatch_group_async(group, tasks[i]->queue, ^{
        rvals[i] = do_something(tasks[i]);
    });
}
```

```
dispatch_group_wait(group, DISPATCH_TIME_FOREVER);
dispatch_release(group);
```

```
// use rvals[]
```

DISPATCH_TIME_FOREVER ?

Time

Time

- Opaque type
  - Not safe for arithmetic
  - Not safe for comparisons
- Two well-known constants
  - DISPATCH_TIME_NOW
  - DISPATCH_TIME_FOREVER

Time

```
// wait forever (this always returns success)
dispatch_group_wait(group, DISPATCH_TIME_FOREVER);
```

```
// poll the group
if (dispatch_group_wait(group, DISPATCH_TIME_NOW)) {
    printf(“Group is not empty\n”);
}
```

Time

- Arbitrary timeouts are supported
- Also possible to defer block execution
  - dispatch_after()

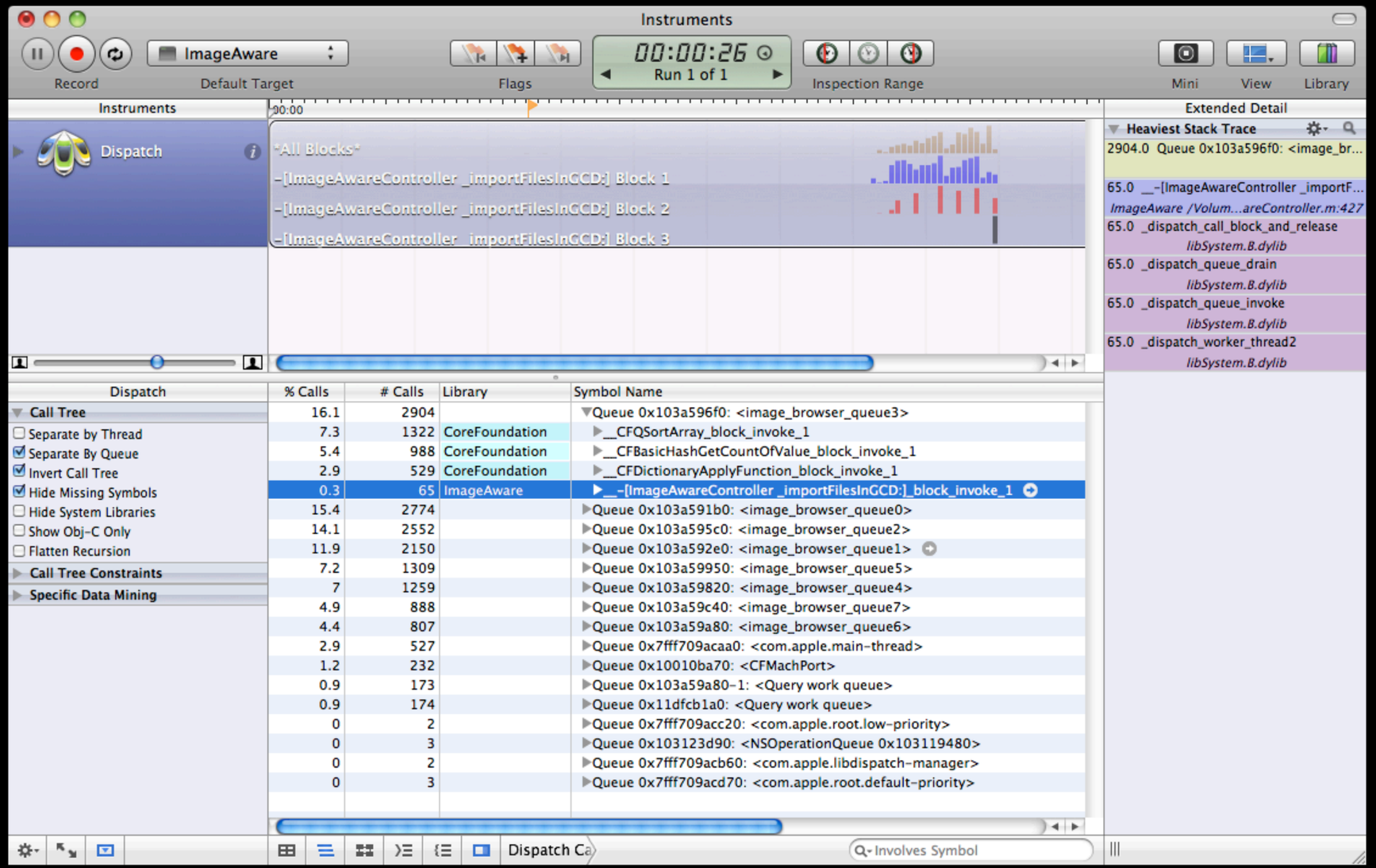
Understanding Grand Central Dispatch in Depth	Presidio Thursday 9:00AM

Instruments

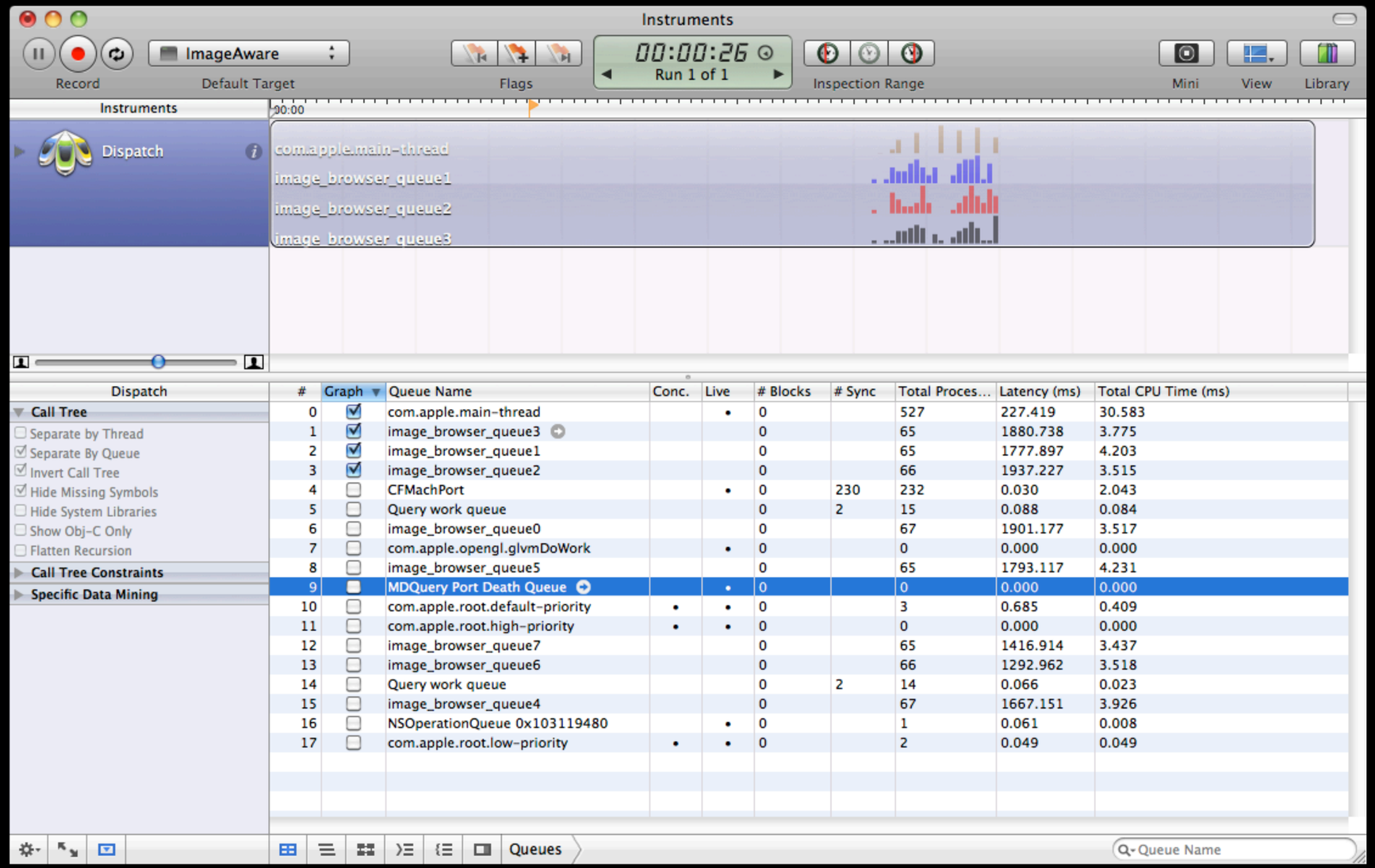
Instruments

- Behavior
  - Latency
  - Track which blocks are enqueued on which queues
  - Which blocks are executed synchronously
- Optimization
  - Longest executing blocks
  - Most executing blocks

Call Tree View

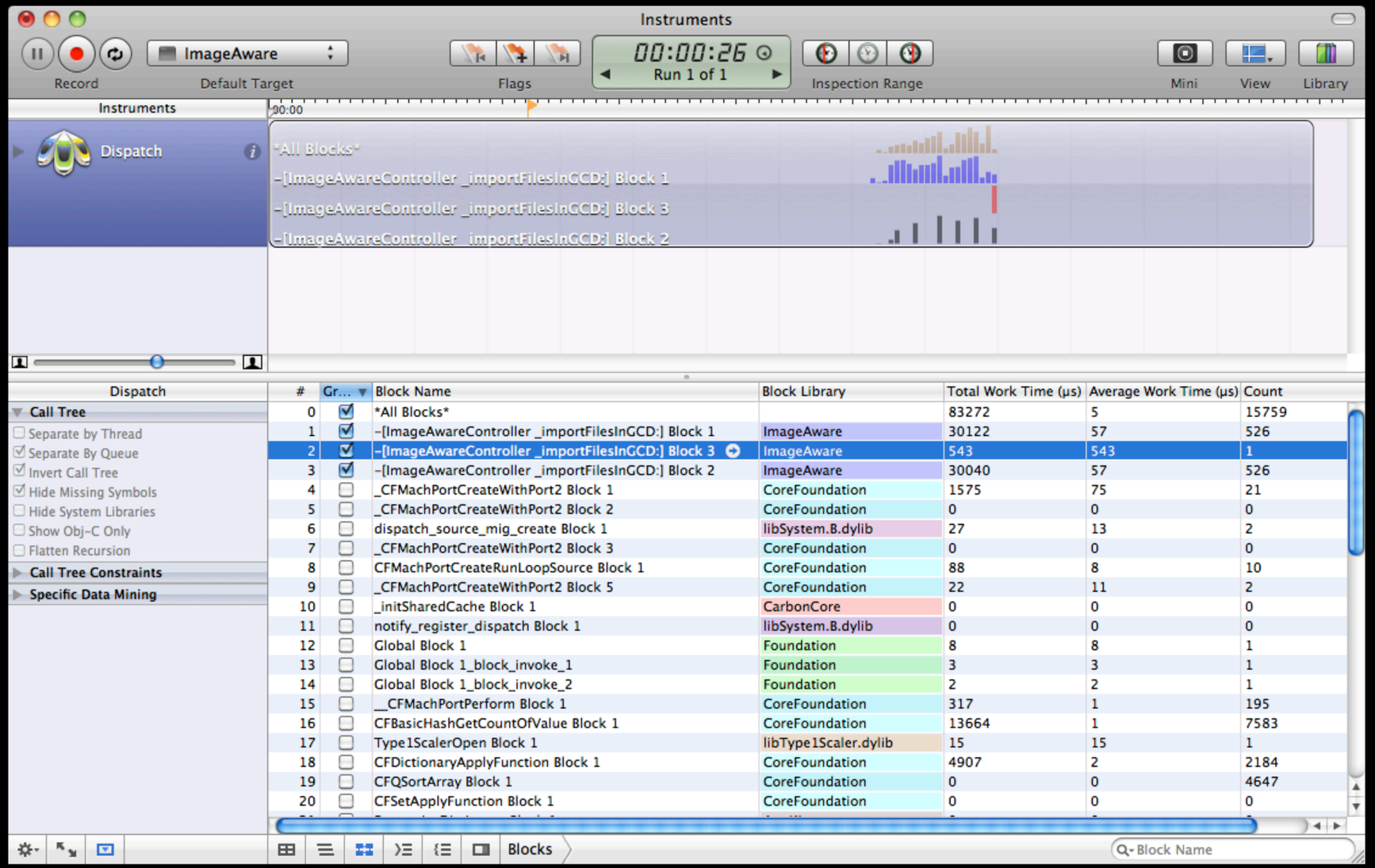


Queue View



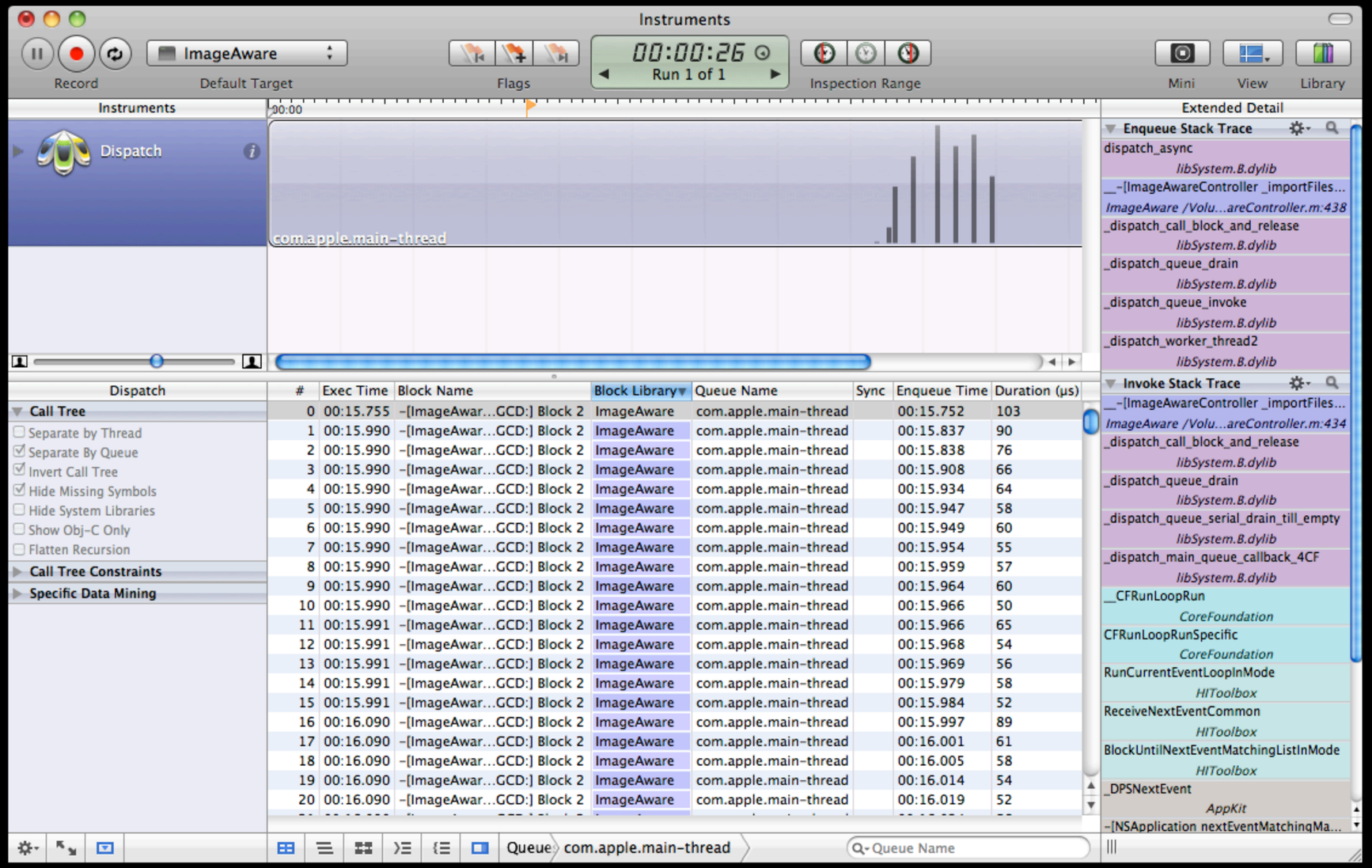
#	Graph	Queue Name	Conc.	Live	# Blocks	# Sync	Total Proces...	Latency (ms)	Total CPU Time (ms)
0	☑	com.apple.main-thread		•	0		527	227.419	30.583
1	☑	image_browser_queue3			0		65	1880.738	3.775
2	☑	image_browser_queue1			0		65	1777.897	4.203
3	☑	image_browser_queue2			0		66	1937.227	3.515
4	☐	CFMachPort		•	0	230	232	0.030	2.043
5	☐	Query work queue			0	2	15	0.088	0.084
6	☐	image_browser_queue0			0		67	1901.177	3.517
7	☐	com.apple.opengl.glvmDoWork		•	0		0	0.000	0.000
8	☐	image_browser_queue5			0		65	1793.117	4.231
9	☐	MDQuery Port Death Queue		•	0		0	0.000	0.000
10	☐	com.apple.root.default-priority	•	•	0		3	0.685	0.409
11	☐	com.apple.root.high-priority	•	•	0		0	0.000	0.000
12	☐	image_browser_queue7			0		65	1416.914	3.437
13	☐	image_browser_queue6			0		66	1292.962	3.518
14	☐	Query work queue			0	2	14	0.066	0.023
15	☐	image_browser_queue4			0		67	1667.151	3.926
16	☐	NSOperationQueue 0x103119480		•	0		1	0.061	0.008
17	☐	com.apple.root.low-priority	•	•	0		2	0.049	0.049

Block View



#	Gr...	Block Name	Block Library	Total Work Time (µs)	Average Work Time (µs)	Count
0	☑	*All Blocks*		83272	5	15759
1	☑	-[ImageAwareController _importFilesInGCD:] Block 1	ImageAware	30122	57	526
2	☑	-[ImageAwareController _importFilesInGCD:] Block 3	ImageAware	543	543	1
3	☑	-[ImageAwareController _importFilesInGCD:] Block 2	ImageAware	30040	57	526
4	☐	_CFMachPortCreateWithPort2 Block 1	CoreFoundation	1575	75	21
5	☐	_CFMachPortCreateWithPort2 Block 2	CoreFoundation	0	0	0
6	☐	dispatch_source_mig_create Block 1	libSystem.B.dylib	27	13	2
7	☐	_CFMachPortCreateWithPort2 Block 3	CoreFoundation	0	0	0
8	☐	CFMachPortCreateRunLoopSource Block 1	CoreFoundation	88	8	10
9	☐	_CFMachPortCreateWithPort2 Block 5	CoreFoundation	22	11	2
10	☐	_initSharedCache Block 1	CarbonCore	0	0	0
11	☐	notify_register_dispatch Block 1	libSystem.B.dylib	0	0	0
12	☐	Global Block 1	Foundation	8	8	1
13	☐	Global Block 1_block_invoke_1	Foundation	3	3	1
14	☐	Global Block 1_block_invoke_2	Foundation	2	2	1
15	☐	__CFMachPortPerform Block 1	CoreFoundation	317	1	195
16	☐	CFBasicHashGetCountOfValue Block 1	CoreFoundation	13664	1	7583
17	☐	Type1ScalerOpen Block 1	libType1Scaler.dylib	15	15	1
18	☐	CFDictionaryApplyFunction Block 1	CoreFoundation	4907	2	2184
19	☐	CFQSortArray Block 1	CoreFoundation	0	0	4647
20	☐	CFSetApplyFunction Block 1	CoreFoundation	0	0	0

Blocks Focus View



Instruments

What's New in Instruments	Pacific Heights Wednesday 3:30PM
Advanced Debugging and Performance Analysis	Nob Hill Thursday 9:00AM

GCD

Fast

Easy

Fun

More Information

Michael Jurewitz
Developer Tools and Performance Evangelist
jurewitz@apple.com

Developer Forums
> Mac Development > Grand Central Dispatch
http://devforums.apple.com/community/mac/gcd

Documentation
Concurrency Programming Guide
http://developer.apple.com/

Related Sessions

Understanding Grand Central Dispatch in Depth	Presidio Thursday 9:00AM
Migrating Your Application to Grand Central Dispatch	Pacific Heights Thursday 2:00PM
Designing Your Cocoa Application for Concurrency	Nob Hill Tuesday 3:30PM
What's New in Instruments	Pacific Heights Wednesday 3:30PM
Objective-C and Garbage Collection Advancements	Russian Hill Wednesday 5:00PM
Advanced Debugging and Performance Analysis	Nob Hill Thursday 9:00AM

Labs

Grand Central Dispatch Lab	Snow Leopard Lab Wednesday 3:30PM
Grand Central Dispatch Lab	Snow Leopard Lab Thursday 3:30PM

Q&A

WWDC Session Survey

You can rate your sessions on the WWDC Attendee Site.

Survey forms are located on each session description page at:

developer.apple.com/wwdc/attendee

WWDC09